

---

# ASYNCHRONOUS JS

PHÓ NGHĨA VĂN

---



**CYBERSOFT**  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



**MYC**  **DER**

**timviec**  **it.com**

# PROJECT: WEATHER APP

Run lệnh tại terminal:

```
node app-promise.js -a "459 su van hanh"
```

Thu được kết quả sau:

Kết quả trả về từ **google API**

```
The address: 459 Sư Vạn Hạnh, Phường 12, Quận 10, Hồ Chí Minh, Vietnam
```

```
=====
Summary: Humid and Partly Cloudy
Icon: partly-cloudy-day
Temperature: 89.48
```

Kết quả trả về từ **darksky API**



# KIẾN THỨC

- Asynchronus
- Callback function
- Callstack và event loop
- Promise và Promise chain
- Google API, Weather API (darksky API)
- Package: request, axios, yargs



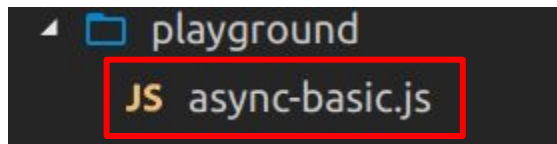
# CÔNG CỤ

- Visual studio code
- Terminal
- NodeJS



# ASYNCHRONOUS

Tạo cấu trúc thư mục và nội dung file **async-basic.js** như sau



\* Folder **playground** chỉ dùng để demo kiến thức, chứ không liên quan trực tiếp đến project

```
playground > JS async-basic.js
1 console.log("The first line")
2
3 setTimeout(() => {
4   console.log("The second line")
5 }, 2000);
6
7 console.log("The third line")
```

Chạy lệnh `node playground/async-basic`, thứ tự các lệnh được in ra ntn?

# ASYNCHRONOUS

Kết quả: dòng console.log **The second line** được thực thi sau **The third line**

```
The first line  
The third line  
The second line
```

Điều này cho thấy: Trong một chuỗi các hàm của một quy trình có n tác vụ, được thực thi theo cơ chế **Async** thì có nghĩa là cho dù hàm B được gọi sau hàm A nhưng không đảm bảo rằng hàm A sẽ phải kết thúc trước hàm B và hàm B bắt buộc phải chỉ được gọi chạy khi hàm A kết thúc

# ASYNCHRONOUS

Tuy nhiên, khi thay đổi thời gian của **setTimeout** thành **zero** thì kết quả thu được vẫn là **The second line** được thực thi sau !!!

```
playground ▶ JS async-basic.js
1 console.log("The first line")
2
3 setTimeout(() => {
4   console.log("The second line")
5 }, 0);
6
7 console.log("The third line")
8
```



```
The first line
The third line
The second line
```



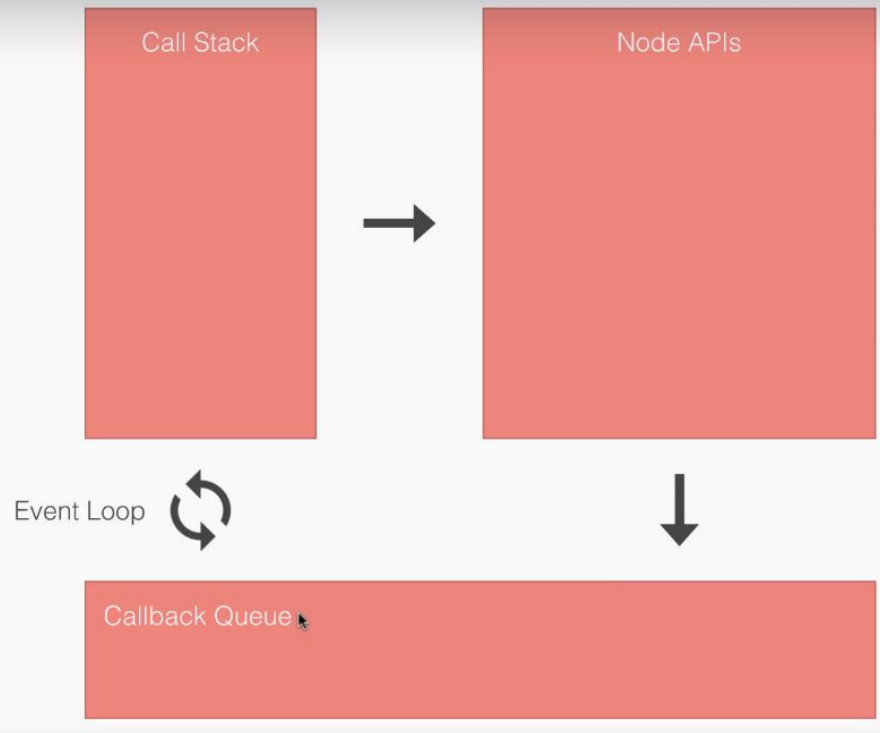
# BEHIND THE SCENE





# CALLSTACK

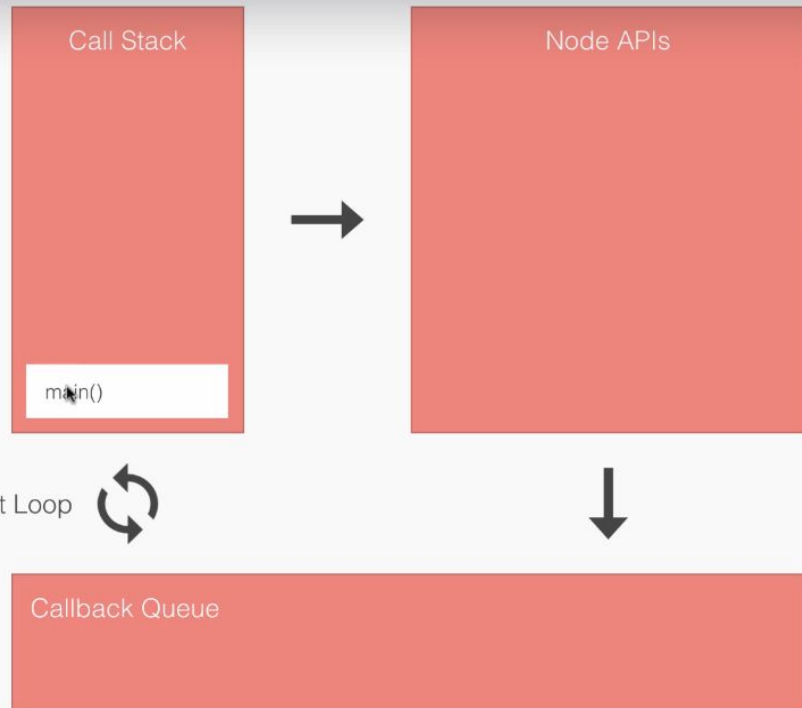
```
var x = 1;  
var y = x + 9;  
console.log(`y is ${y}`);
```



# CALLSTACK

Đầu tiên chúng ta có  
function **main()**

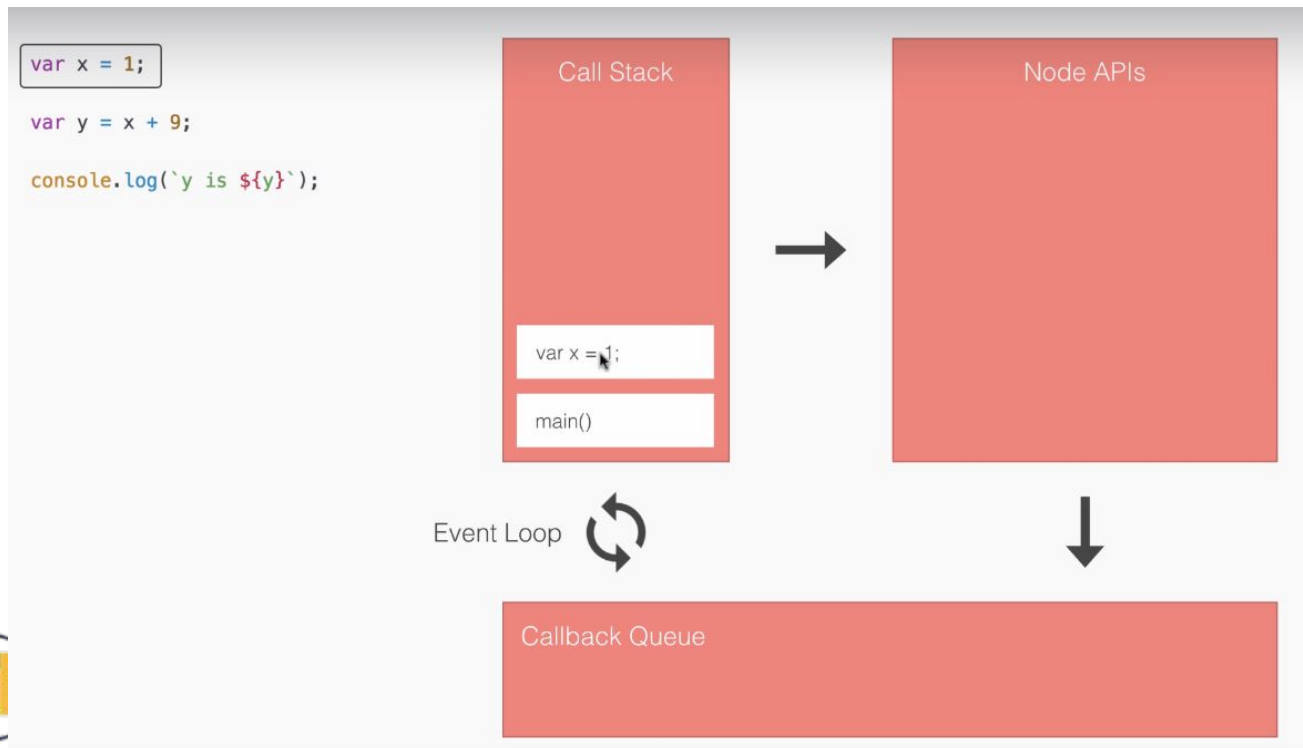
```
var x = 1;  
var y = x + 9;  
console.log(`y is ${y}`);
```



# CALLSTACK

**var x = 1;** nằm trên  
main()

Sau khi thực thi, lệnh  
trên sẽ được remove

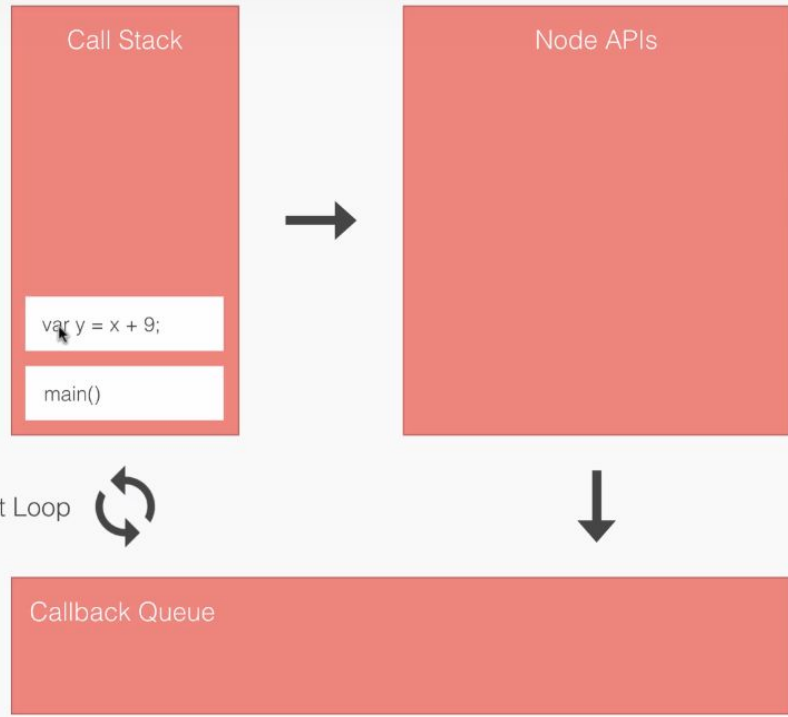


# CALLSTACK

Lệnh thứ 2 được thực hiện: **var y = x + 9;**

Sau đó cũng sẽ được xóa đi

```
var x = 1;  
var y = x + 9;  
console.log(`y is ${y}`);
```

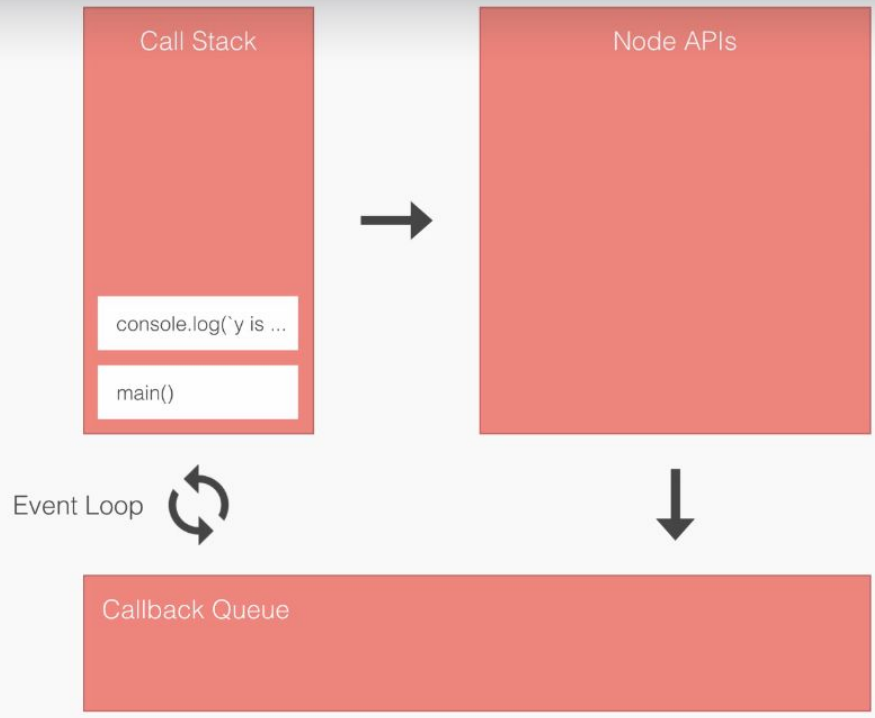


# CALLSTACK

Tương tự đối với lệnh  
**Console.log**

Được thực thi, sau đó  
được xóa đi

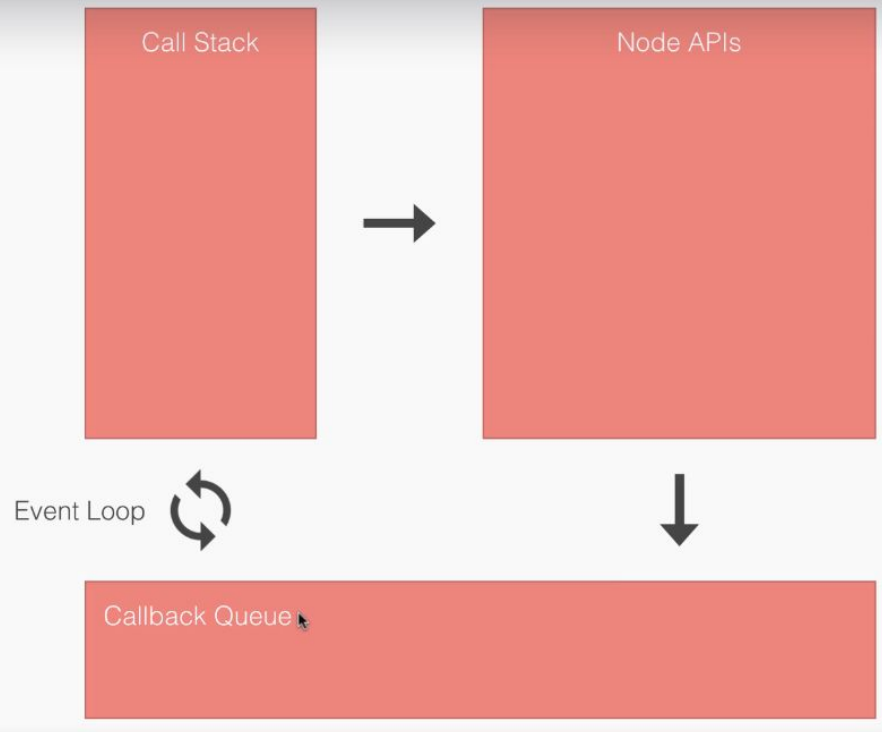
```
var x = 1;  
var y = x + 9;  
console.log(`y is ${y}`);
```



# CALLSTACK

Cuối cùng, hàm **main()** cũng thực thi xong và cũng được xóa.  
Chương trình kết thúc.

```
var x = 1;  
var y = x + 9;  
console.log(`y is ${y}`);
```



# CALLSTACK

```
console.log('Starting app');

setTimeout(() => {
  console.log('Inside of callback');
}, 2000);

setTimeout(() => {
  console.log('Second setTimeout');
}, 0);

console.log('Finishing up');
```

Call Stack

Node APIs

Event Loop



Callback Queue

# CALLSTACK

```
console.log('Starting app');

setTimeout(() => {
  console.log('Inside of callback');
}, 2000);

setTimeout(() => {
  console.log('Second setTimeout');
}, 0);

console.log('Finishing up');
```

Call Stack

console.log('Star...

main()

Node APIs

Event Loop

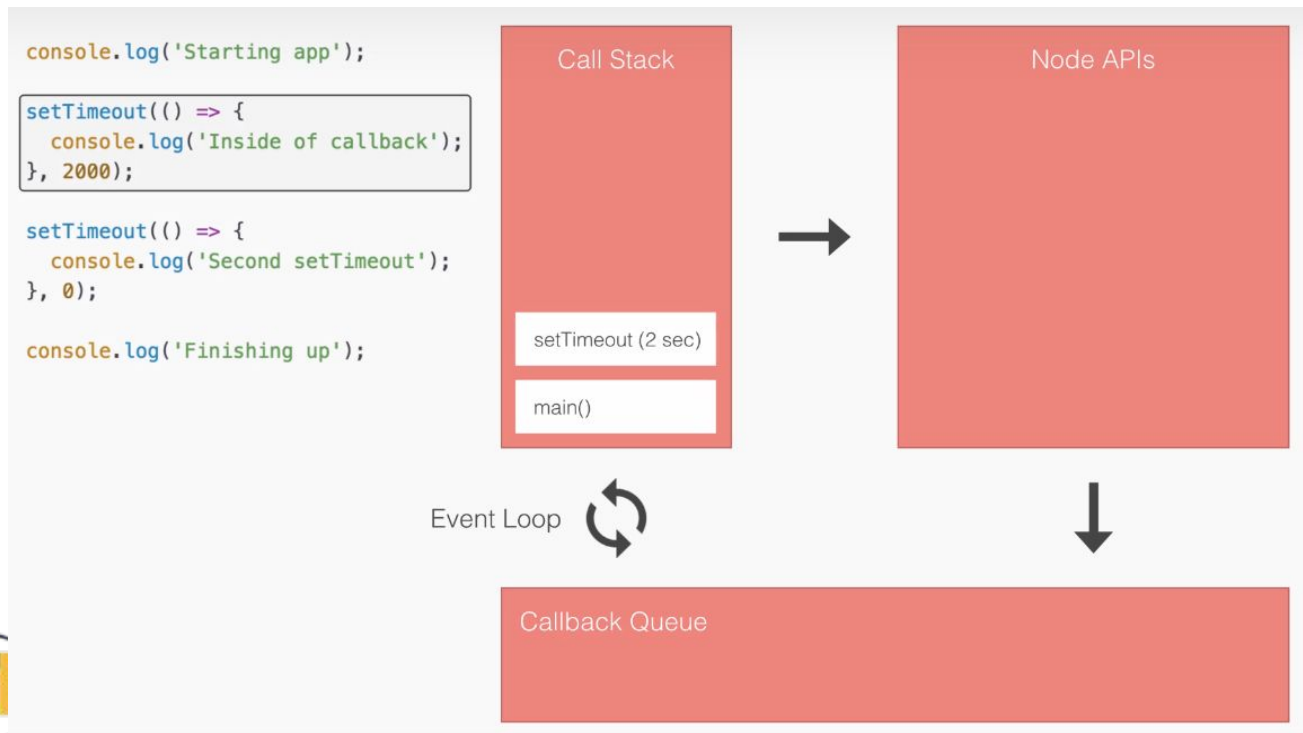


Callback Queue



# CALLSTACK

**setTimeout** không thuộc V8 engine, mà thuộc Node API nên khi được thực thi, sẽ được register tại NodeAPI sau thời gian **timeout**



# CALLSTACK

```
console.log('Starting app');  
  
setTimeout(() => {  
  console.log('Inside of callback');  
}, 2000);  
  
setTimeout(() => {  
  console.log('Second setTimeout');  
}, 0);  
  
console.log('Finishing up');
```

Call Stack

setTimeout (2 sec)

main()

Node APIs

setTimeout (2 sec)

Event Loop



Callback Queue

# CALLSTACK

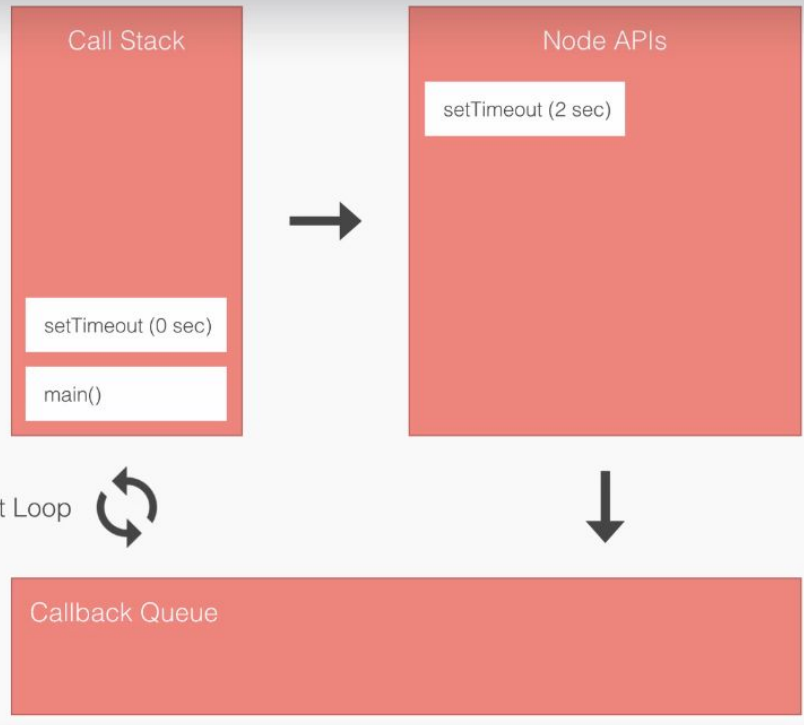
Khi **setTimeout 1** được thực thi tại **Node APIs** thì **setTimeout 2** được gọi và thực thi tại **Callstack**

```
console.log('Starting app');

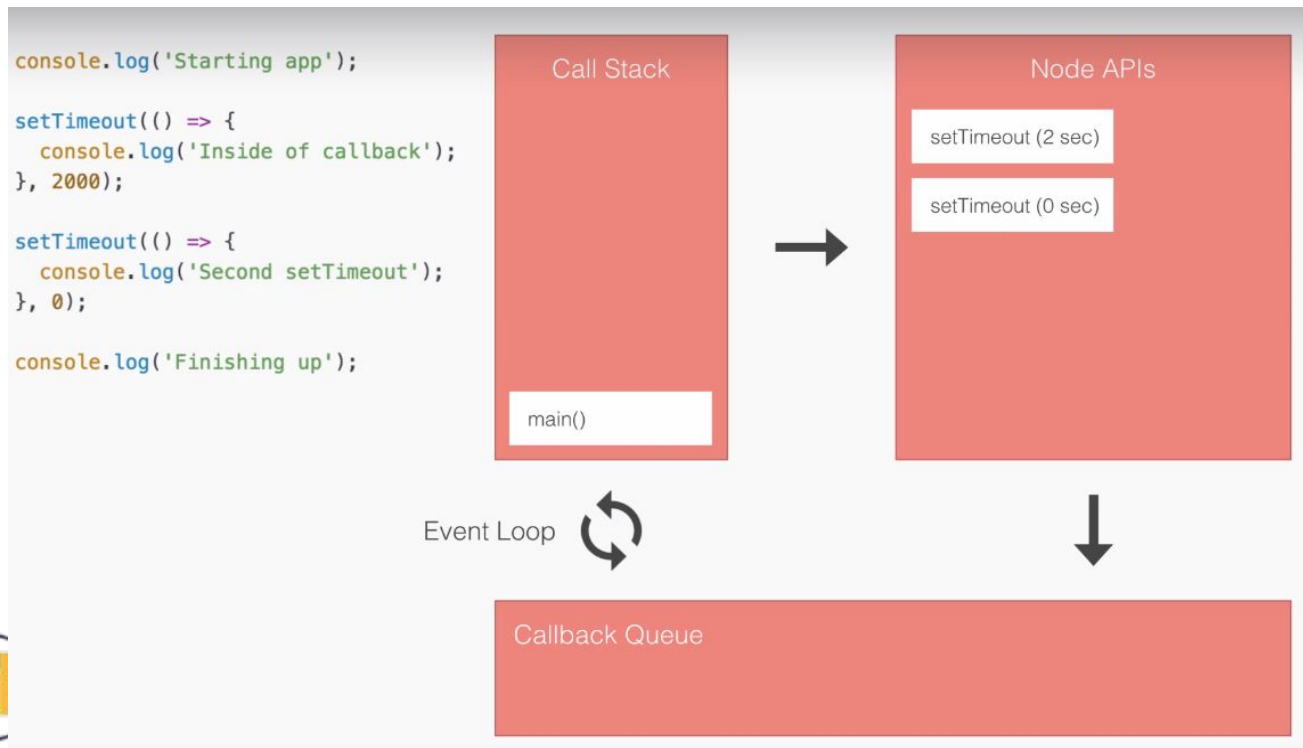
setTimeout(() => {
  console.log('Inside of callback');
}, 2000);

setTimeout(() => {
  console.log('Second setTimeout');
}, 0);

console.log('Finishing up');
```



# CALLSTACK

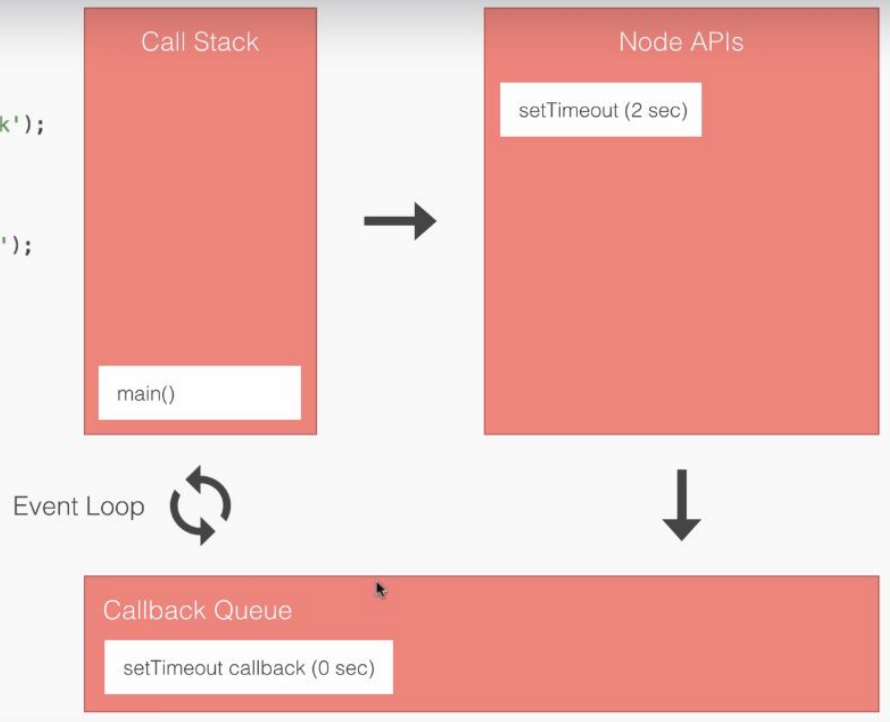


# CALLSTACK

**setTimeout 2** được đưa đến **Callback queue** trước, vì **timeout** ngắn hơn.

**Callback queue** là nơi chờ đợi tất cả các **callback** function. Đợi đến khi Callstack **empty** thì có thể thực thi tuần

```
console.log('Starting app');  
  
setTimeout(() => {  
  console.log('Inside of callback');  
}, 2000);  
  
setTimeout(() => {  
  console.log('Second setTimeout');  
}, 0);  
  
console.log('Finishing up');
```



**CYBERSOFT**  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



# CALLSTACK

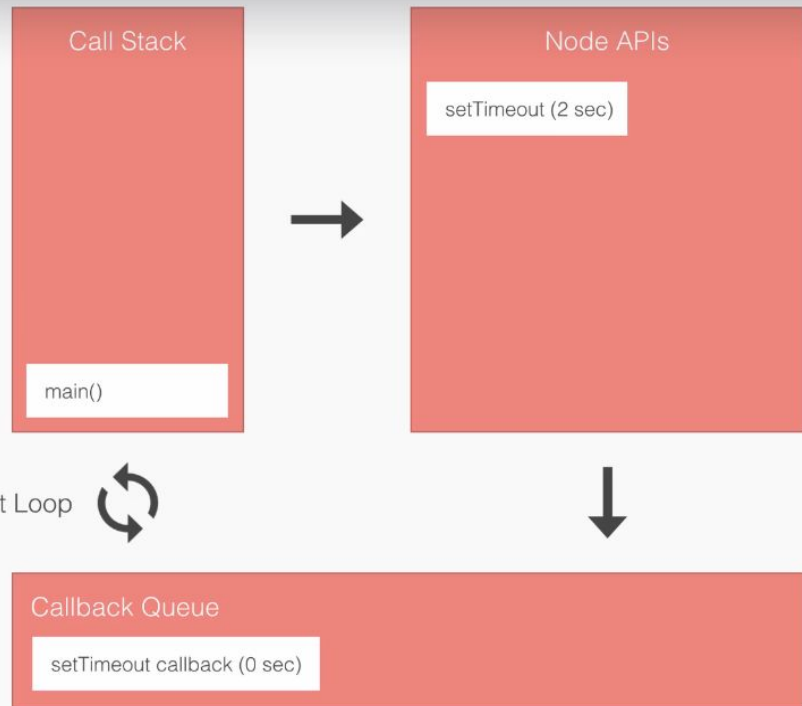
console.log **Finishing up** được thực thi trong **Callstack**

```
console.log('Starting app');

setTimeout(() => {
  console.log('Inside of callback');
}, 2000);

setTimeout(() => {
  console.log('Second setTimeout');
}, 0);

console.log('Finishing up');
```



# CALLSTACK

Hàm **main()** thực thi xong và được xóa khỏi **Callstack**.

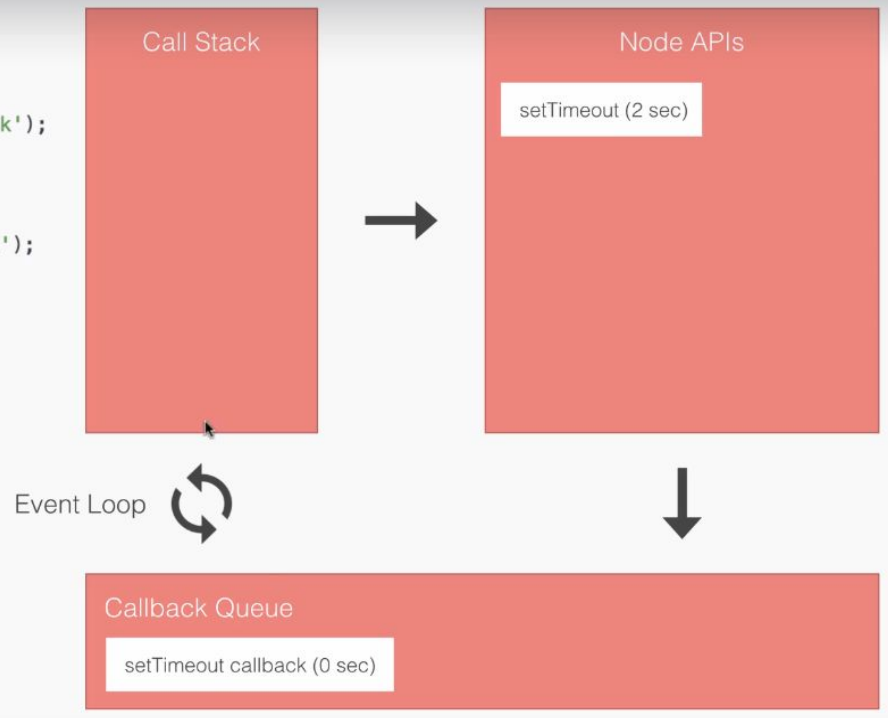
Sau đó, **setTimeout 2** được đưa vào **Callstack** và thực thi

```
console.log('Starting app');

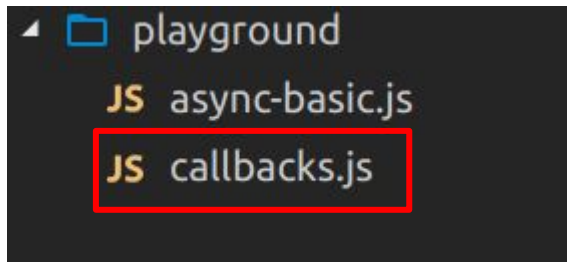
setTimeout(() => {
  console.log('Inside of callback');
}, 2000);

setTimeout(() => {
  console.log('Second setTimeout');
}, 0);

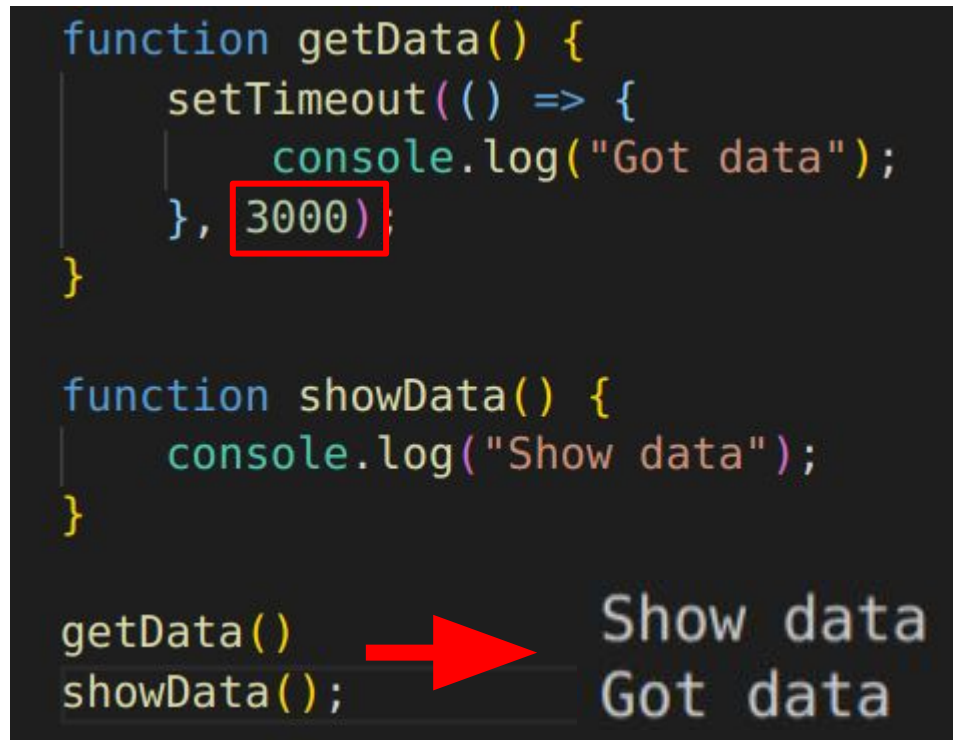
console.log('Finishing up');
```



# CALLBACK FUNCTION



Trong thực tế, quá trình lấy data có thể mất một khoảng thời gian như ví dụ bên. Do đó kết quả sẽ console log ra **Show data** trước, mặc dù ta chưa có data để show





# CALLBACK FUNCTION

**Callback function** là một function được truyền như một parameter vào bên trong một hàm khác.

**callback** có vai trò như một **người giám sát**, có nhiệm vụ giám sát xem khi nào thực hiện xong task **getData** thì mới thực hiện tiếp task **showData**

--> Giúp tránh được **Asynchronus**



```
function getData(callback) {  
  setTimeout(function() {  
    console.log("Got data");  
    callback();  
  }, 3000);  
}
```

```
function showData() {  
  console.log("Show data");  
}
```

```
getData(() => {  
  showData();  
});
```

→ Got data  
Show data

# GEO-LOCATION - GOOGLE APIs



# GEO-LOCATION

Nhập vào địa chỉ sau trên thanh URL của trình duyệt

 [https://maps.googleapis.com/maps/api/geocode/json?key=\[key\]&address=459 Sư Vạn Hạnh](https://maps.googleapis.com/maps/api/geocode/json?key=[key]&address=459 Sư Vạn Hạnh)

Nhập vào **google key** (tự đăng ký hoặc sử dụng key của trung tâm)

Nhập vào địa chỉ (có thể viết tiếng Việt, có dấu và khoảng trống)

Cài đặt extension **JSONView** trong google Chrome để hiện thị kết quả dạng json



# GEO-LOCATION

Kết quả trả về như sau:

```
formatted_address: "459 Sư Vạn Hạnh, Phường 12, Quận 10, Hồ Chí Minh, Vietnam",
- geometry: {
  - location: {
    lat: 10.7732835,
    lng: 106.6686639
  },
  location_type: "RANGE_INTERPOLATED",
  - viewport: {
    - northeast: {
      lat: 10.7746324802915,
      lng: 106.6700128802915
    },
    - southwest: {
      lat: 10.7719345197085,
      lng: 106.6673149197085
    }
  },
  place_id: "Ekk0NTkgU8awIFbhugFuIEjhuqFuaCwgUGjGs0G7nW5nIDeYLCBRdeG6rW4gMTAsIEjhu5Mg
- types: [
  "street_address"
]
},
status: "OK"
```

2

```
{
  - results: [
    - {
      - address_components: [
        - {
          long_name: "459",
          short_name: "459",
          - types: [
            "street_number"
          ]
        },
        - {
          long_name: "Sư Vạn Hạnh",
          short_name: "Sư Vạn Hạnh",
          - types: [
            "route"
          ]
        },
        - {
          long_name: "Quận 10",
          short_name: "Quận 10",
          - types: [
            "administrative_area_level_2",
            "political"
          ]
        },
        - {
          long_name: "Hồ Chí Minh",
          short_name: "Hồ Chí Minh",
          - types: [
            "administrative_area_level_1",
            "political"
          ]
        },
        - {
          long_name: "Vietnam",
          short_name: "VN",
          - types: [
            "country",
            "political"
          ]
        }
      ]
    }
  ]
}
```

1

# VÀO PROJECT THÔI !!!



# KHỞI TẠO PROJECT

- Run lệnh **npm init -y** trong terminal/commnad prompt (**-y**: sử dụng mặc định, không khai báo thêm thông tin gì)
- Run lệnh **npm install**
- Run lệnh **npm install request** để sử dụng package **request**. Package này hỗ trợ gửi **http request**



# REQUEST

- Tạo file index.js
- Gọi package **request**
- Gọi hàm request với các tham số như hình

```
playground
  JS async-basic.js
  JS callbacks.js
  JS index.js
  npm package-lock.json
  npm package.json
```

```
const request = require('request');

request({
  url: 'https://maps.googleapis.com/maps/api/geocode/json?key=
  json: true
},
  (err, res, body) => {
    console.log(body)
  }
})
```



# REQUEST

Run **node index** trên terminal sẽ được kết quả như trên browser

```
hackagon@TheMeta: /media/hackagon/MetaData/CybersoftAcademy/NodeJS_Lecture/Project-weather-app/weather-app$ node index
{ results:
  [ { address_components: [Array],
    formatted_address: '459 Sư Vạn Hạnh, Phường 12, Quận 10, Hồ Chí Minh, Vietnam',
    geometry: [Object],
    place_id: 'Ekk0NTkgU8awIFbhuqFuIEjhuqFuaCwgUGjGsOG7nW5nIDEyLCBRdeG6rW4gMTAsIEjhu5MgQ2jDrSBNaw5oLCBWaeG7h3QgTmFtI
Qywm',
    types: [Array] } ],
  status: 'OK' }
```





# REQUEST

\* **Chú ý:** Kết quả hiển thị **[Object]/[Array]** thay vì in nội dung của object --> có thể sử dụng `JSON.stringify(...)`

```
const request = require('request');

request({
  url: 'https://maps.googleapis.com/maps/api/geocode/json?key=...',
  json: true
}, (err, res, body) => {
  console.log(JSON.stringify(body, undefined, 2))
})
```



# REQUEST

hackagon@TheMeta: /media/hackagon/MetaData/CybersoftAcademy/NodeJS\_Lecture/Project-weather-app/weather-app\$ node index

```
{
  "results": [
    {
      "address_components": [
        {
          "long_name": "459",
          "short_name": "459",
          "types": [
            "street_number"
          ]
        }
      ],
      {
        "long_name": "Sư Vạn Hạnh",
        "short_name": "Sư Vạn Hạnh",
        "types": [
          "route"
        ]
      },
      {
        "long_name": "Quận 10",
        "short_name": "Quận 10",
        "types": [
          "administrative_area_level_2",
          "political"
        ]
      },
    ],
  }
}
```

Đã in được kết quả bên trong array/object



# REQUEST

In những thông tin cần thiết

```
const request = require('request');

request({
  url: 'https://maps.googleapis.com/maps/api/geocode/json?key=AIzaSyCpf',
  json: true
},
(err, res, body) => {
  console.log(`Address: ${body.results[0].formatted_address}`);
  console.log(`Latitude: ${body.results[0].geometry.location.lat}`);
  console.log(`Longitude: ${body.results[0].geometry.location.lng}`);
})
```

hackagon@TheMeta: /media/hackagon/MetaData/CybersoftAcademy/NodeJS\_Lecture/Project-weather-app/weather-app\$ node index

Address: 459 Sư Vạn Hạnh, Phường 12, Quận 10, Hồ Chí Minh, Vietnam

Latitude: 10.7732835

Longitude: 106.6686639

# YARGS

- `npm install yargs`
- Package **yargs** giúp xây dựng ứng dụng có tính tương tác cao và thân thiện với người dùng



```
const request = require('request');
const yargs = require('yargs');

const argv = yargs
  .options({
    a: {
      demand: true,
      alias: 'address',
      describe: 'Enter your target address',
      string: true
    }
  })
  .help()
  .alias('help', 'h')
  .argv;
```

viết tắt của address

Trợ giúp trong terminal

Chấp nhận các cài đặt, lưu trữ kết quả tại argv

# YARGS

Run lệnh **node index -h** hoặc **node index --help**

```
hackagon@TheMeta: /media/hackagon/MetaData/CybersoftAcademy/NodeJS_Lecture/Project-weather-app/weather-app$ node index -h
```

```
Options:
  --version      Show version number                [boolean]
  -a, --address  Enter your target address           [string] [required]
  --help, -h     Show help                          [boolean]
```



# YARGS

- `console.log(argv)`
- Run lệnh `node index -a "459 Su Van Hanh"` hoặc `node index --address "459 Su Van Hanh"` thu được kết quả như sau:

```
/weather-app$ node index -a "459 Su Van Hanh"
{ _: [],
  a: '459 Su Van Hanh',
  address: '459 Su Van Hanh',
  '$0': 'index' }
```

# ENCODING USER INPUT

node\_modules

playground

JS async-basic.js

JS callbacks.js

JS encode-decode.js

JS index.js

package-lock.json

package.json

```
// Encoding Mã hóa tất cả những ký tự đặc biệt có trong chuỗi URI
let input = '459 Su Van Hanh';
console.log('Encode: ' + encodeURIComponent(input))
```

```
// Decoding
let encodedInput = '459%20Su%20Van%20Hanh'
console.log('Decode: ' + decodeURIComponent(encodedInput))
```



CYBERSOFT  
ĐÀO TẠO CHUYÊN NGHIỆP

```
hackagon@TheMeta: /media/hackagon/MetaData/CybersoftAcademy/NodeJS_Lecture/Project-weather-app/weather-app$ node ./playground/encode-decode
```

```
Encode: 459%20Su%20Van%20Hanh
Decode: 459 Su Van Hanh
```



# ENCODING USER INPUT

```
let encodedAddress = encodeURIComponent(argv.address);
```

```
request({  
  url: `https://maps.googleapis.com/maps/api/geocode/json?  
  key=[REDACTED]&address=${encodedAddress}`,  
  json: true  
},
```

```
hackagon@TheMeta: /media/hackagon/MetaData/CybersoftAcademy/NodeJS_Lecture/  
r-app/weather-app$ node index --address "82 Ung van Khiem"  
Address: 82 Ung Văn Khiêm, Phường 25, Bình Thạnh, Hồ Chí Minh, Vietnam  
Latitude: 10.8074551  
Longitude: 106.7167241
```



# HANDLE ERRORS

- Error 1: không thể kết nối đến google server
- Error 2: Khi sử dụng zip code 0000 (không tồn tại) trên thanh URL của trình duyệt sẽ được kết quả như hình --> Chúng ta quan tâm đến **status**
- Success: **status: "OK"**

&address=0000



```
{  
  results: [ ],  
  status: "ZERO_RESULTS"  
}
```

# HANDLE ERRORS

- Sử dụng **if else** để xử lý errors
- Test ...

```
(err, res, body) => {  
  if (err) {  
    console.log('Unable to connect to Google server');  
  } else if (body.status === "ZERO_RESULTS") {  
    console.log('Address Not Found');  
  } else if (body.status === "OK") {  
    console.log(`Address: ${body.results[0].formatted_address}`);  
    console.log(`Latitude: ${body.results[0].geometry.location.lat}`);  
    console.log(`Longitude: ${body.results[0].geometry.location.lng}`);  
  }  
}
```

# GEOCODE MODULE

Đóng gói request geocode thành module như sau:

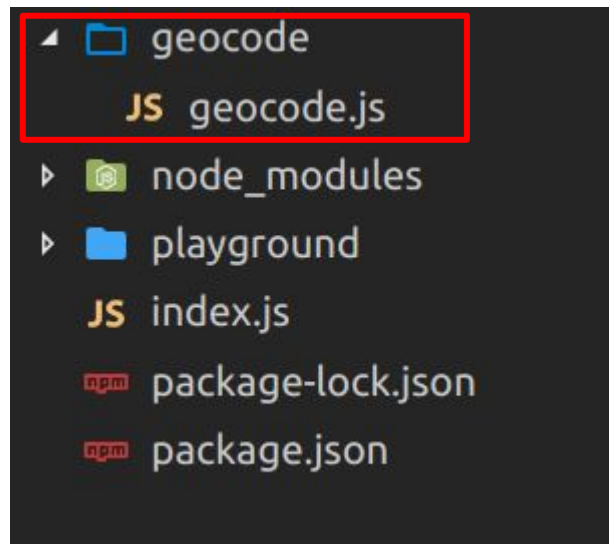
- Tạo cấu trúc thư mục như hình
- Viết module, xuất module, nhập module

```
const geocodeAddress = (address) => {  
}  
module.exports.geocodeAddress = geocodeAddress;  
...
```

viết trong geocode/geocode.js

```
const geocode = require('./geocode/geocode');
```

```
geocode.geocodeAddress(argv.address);
```



# GEOCODE MODULE

Cut and paste từ file index.js là xong :))  
có thể in kết quả ra ngoài terminal

Tuy nhiên, nếu viết như vậy, thì chúng ta chỉ in được kết quả, chứ không thể lấy kết quả đó (latitude và longitude) để gọi api weather --> do đó cần phải sử dụng **callback function**.



```
const request = require('request');

const geocodeAddress = (address) => {
  let encodedAddress = encodeURIComponent(address);

  request(
    {
      url: `https://maps.googleapis.com/maps/api/geocode/json?address=${encodedAddress}`,
      json: true
    },
    (err, res, body) => {
      if (err) {
        console.log('Unable to connect to Google Maps API');
      } else if (body.status === "ZERO_RESULTS") {
        console.log('Address Not Found');
      } else if (body.status === "OK") {
        console.log(`Address: ${body.results[0].formatted_address}`);
        console.log(`Latitude: ${body.results[0].geometry.location.lat}`);
        console.log(`Longitude: ${body.results[0].geometry.location.lng}`);
      }
    }
  );
}

module.exports.geocodeAddress = geocodeAddress;
```

# GEOCODE MODULE

```
const geocodeAddress = (address, callback) => {  
  let encodedAddress = encodeURIComponent(address);  
  
  request({  
    {  
      url: `https://maps.googleapis.com/maps/api/geocode/json?key=  
      json: true  
    },  
  
    (err, res, body) => {  
      if (err) {  
        callback('Unable to connect to Google server')  
      } else if (body.status === "ZERO_RESULTS") {  
        callback('Address Not Found');  
      } else if (body.status === "OK") {  
        callback(undefined, {  
          Address: body.results[0].formatted_address,  
          Latitude: body.results[0].geometry.location.lat,  
          Longitude: body.results[0].geometry.location.lng  
        })  
      }  
    }  
  })  
}
```

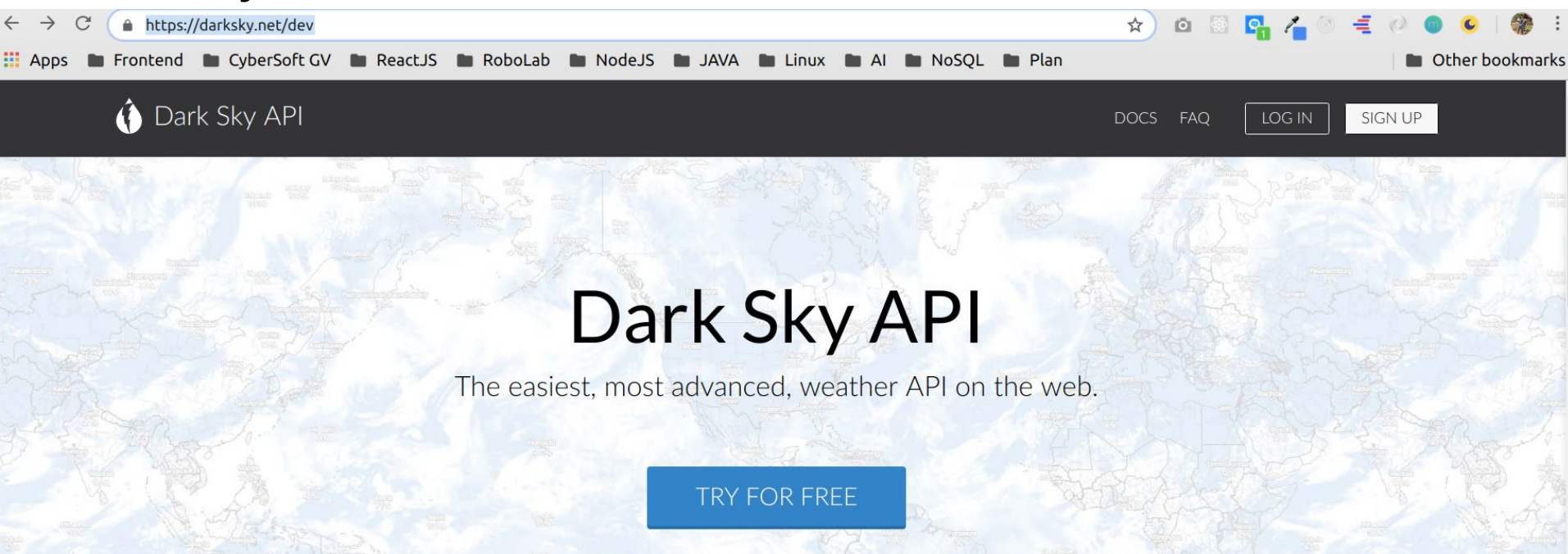
```
geocode.geocodeAddress(argv.address, (err,  
  if(err){  
    console.log(err);  
  } else {  
    console.log(res);  
  }  
})
```





# WEATHER API

Vào trang <https://darksky.net/dev> đăng ký account và login --> được cung cấp **secret key**



# WEATHER API

Test Weather API

 [https://api.darksky.net/forecast/\[redacted\]/37.8267,-122.4233](https://api.darksky.net/forecast/[redacted]/37.8267,-122.4233)

Kết quả trả về **JSON object**

**Secret Key**

**<latitude>,<longtitude>**

# WEATHER API

Viết request để  
lấy data từ  
**darksky**

```
request({  
  url: 'https://api.darksky.net/forecast/  
  json: true  
}, (err, res, body) => {  
  if (err) {  
    console.log('Unable to connect to the Darksky server');  
  } else if (body.code === 400) {  
    console.log('NOT FOUND LOCATION');  
  } else {  
    console.log('Summary:' + body.currently.summary);  
    console.log('Icon:' + body.currently.icon);  
    console.log('Temperature:' + body.currently.temperature);  
  }  
}
```

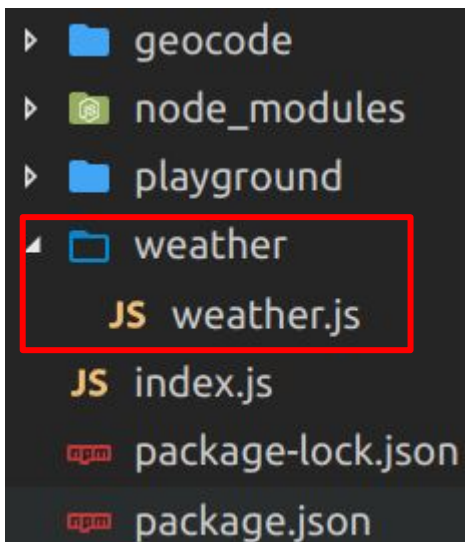


**CYBERSOFT**  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



# WEATHER MODULE

- Tạo cấu trúc thư mục
- Cắt và đưa nội dung vào hàm `getWeather()`



```
const request = require("request");

getWeather = () => {
  request(
    {
      url:
        "https://api.darksky.net/forecast/b8164e",
      json: true
    },
    (err, res, body) => {
      if (err) {
        console.log("Unable to connect to the Da");
      } else if (body.code === 400) {
        console.log("NOT FOUND LOCATION");
      } else {
        console.log("Summary:" + body.currently.);
        console.log("Icon:" + body.currently.ico);
        console.log("Temperature:" + body.curren);
      }
    }
  );
};

module.exports.getWeather = getWeather;
```

# WEATHER MODULE

Làm tương tự geocode, sử dụng **callback function** để có thể trả về một JSON object

Run lệnh **node index -a "abc"** được kết quả như hình dưới

```
weather.getWeather(10.7732835, 10.7732835, (err, res) => {  
  if(err){  
    console.log(err);  
  } else {  
    console.log(JSON.stringify(res, undefined, 2))  
  }  
})
```

```
getWeather = (latitude, longitude, callback) => {  
  request(  
    {  
      url:  
        `https://api.darksky.net/forecast/  
        ${latitude},${longitude}`,  
      json: true  
    },  
    (err, res, body) => {  
      if (err) {  
        callback("Unable to connect to the Darksky server");  
      } else if (body.code === 400) {  
        callback("NOT FOUND LOCATION");  
      } else {  
        callback({  
          Summary: body.currently.summary,  
          Icon: body.currently.icon,  
          Temperature: body.currently.temperature  
        })  
      }  
    })  
  )  
}
```

**Kết quả:** { Summary: 'Clear', Icon: 'clear-day', Temperature: 90.71 }

# PROMISE - ES6

- Để giải quyết vấn đề bất đồng bộ --> sử dụng callback function
- Nếu sử dụng quá nhiều callback function --> **callback hell**
- Sử dụng quá nhiều callback lồng vào nhau --> khó maintain code sau này

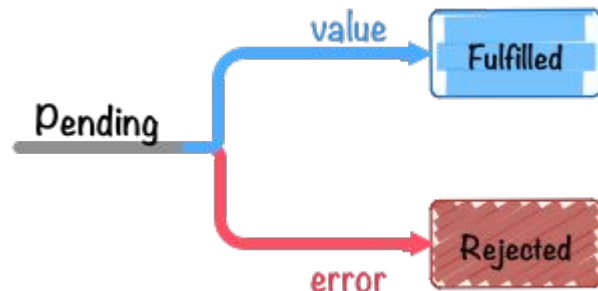
**Promise** ra đời để giải quyết vấn đề này (**promise chaining**)

```
1 function hell(win) {  
2   // for listener purpose  
3   return function() {  
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {  
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {  
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {  
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {  
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {  
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {  
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {  
11                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {  
12                  loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {  
13                   async.eachSeries(SCRIPTS, function(src, callback) {  
14                     loadScript(win, BASE_URL+src, callback);  
15                   });  
16                 });  
17               });  
18             });  
19           });  
20         });  
21       });  
22     });  
23   });  
24 });  
25 }  
26 }
```

# PROMISE - ES6

Promise có 3 trạng thái:

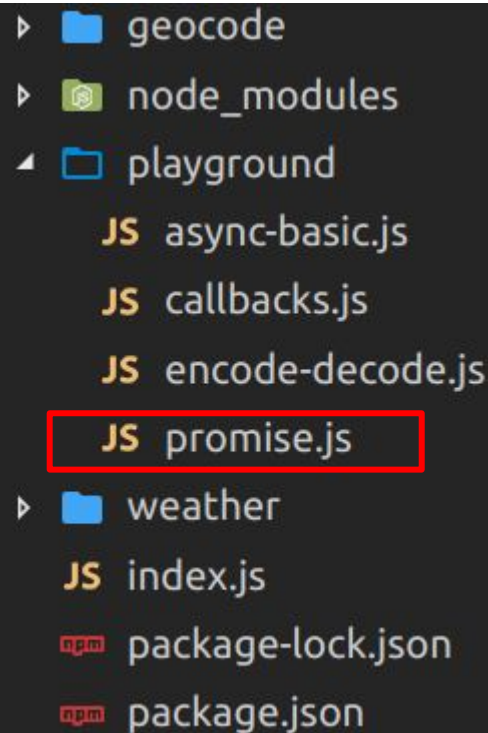
- **Pending:** đang xử lý (bất đồng bộ) để có kết quả cuối cùng. Trạng thái này sẽ chuyển thành một trong hai trạng thái kia
- **Fulfilled:** xảy ra nếu quá trình xử lý trả về kết quả cuối cùng (kể cả undefined)
- **Rejected:** xảy ra khi có lỗi trong quá trình xử lý



# PROMISE - PLAYGROUND

- Tạo file `./playground/promise.js` làm nháp
- Khởi tạo promise với callback function có 2 tham số: **resolve** (xử lý thành công), **reject** (có lỗi xảy ra)
- **setTimeout** tượng trưng cho trạng thái **pending**
- **Lưu ý:** chỉ có thể gọi resolve hoặc reject 1 lần

```
let somePromise = new Promise( resolve, reject ) => {  
  setTimeout(() => {  
    resolve('Hey. It worked');  
    // reject('Hey. It did not work');  
  }, 2000);  
})
```



```
▶ geocode  
▶ node_modules  
└─▶ playground  
    JS async-basic.js  
    JS callbacks.js  
    JS encode-decode.js  
    JS promise.js  
▶ weather  
    JS index.js  
    npm package-lock.json  
    npm package.json
```

# PROMISE - PLAYGROUND

- Phương thức **.then(...)** có 2 tham số: 1 callback cho xử lý **thành công** và 1 callback cho xử lý **thất bại**
- Callback nào được thực thi phụ thuộc vào kết quả xử lý tại Promise dẫn đến trạng thái **fulfilled** hay hay **rejected**.

```
somePromise.then((message) => {  
    console.log('Success: ', message);  
}, (err) => {  
    console.log('Err: ', err);  
})
```



# PROMISE - PLAYGROUND

```
let somePromise = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    resolve('Hey. It worked');  
    // reject('Hey. It did not work');  
  }, 2000);  
})
```

Success: Hey. It worked

```
let somePromise = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    // resolve('Hey. It worked');  
    reject('Hey. It did not work');  
  }, 2000);  
})
```

Err: Hey. It did not work

# PROMISE - PLAYGROUND

Ngoài ra có thể sử dụng 2 phương thức `.then( ... )` và `.catch( ... )` như sau (trả về kết quả tương tự)

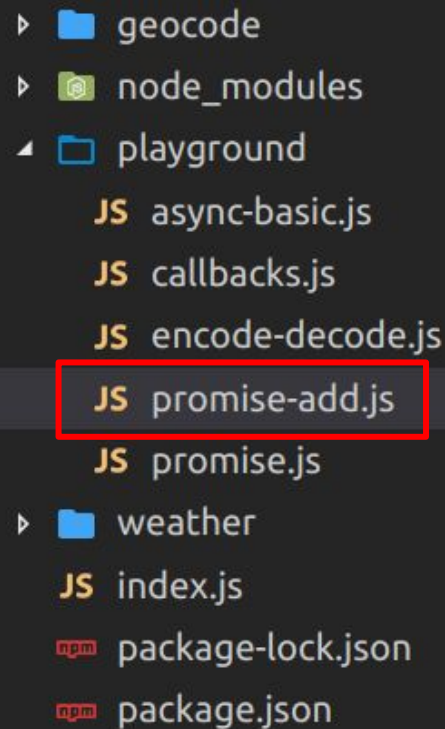
```
somePromise
  .then((message) => {
    console.log('Success: ', message);
  })
  .catch((err) => {
    console.log('Err: ', err);
  })
```



# PROMISE CHAINING - PLAYGROUND

- Trường hợp kết quả trả về của promise này lại là input cho promise tiếp theo.
- VD: ta cần tính  $1 + 2 = 3$ , sau đó  $3 + 4 = 7$  (3 là kết quả của  $1 + 2$ ). Hay  $(1 + 2) + 4 = 7$ .

```
const add = (a, b) => {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      if(typeof a === 'number' && typeof b === 'number'){  
        resolve(a + b);  
      } else {  
        reject('Arguments must be numbers')  
      }  
    }, 1000);  
  })  
}
```



```
▶ folder geocode  
▶ folder node_modules  
◀ folder playground  
  JS async-basic.js  
  JS callbacks.js  
  JS encode-decode.js  
  JS promise-add.js  
  JS promise.js  
▶ folder weather  
  JS index.js  
  npm package-lock.json  
  npm package.json
```

# PROMISE CHAINING - PLAYGROUND

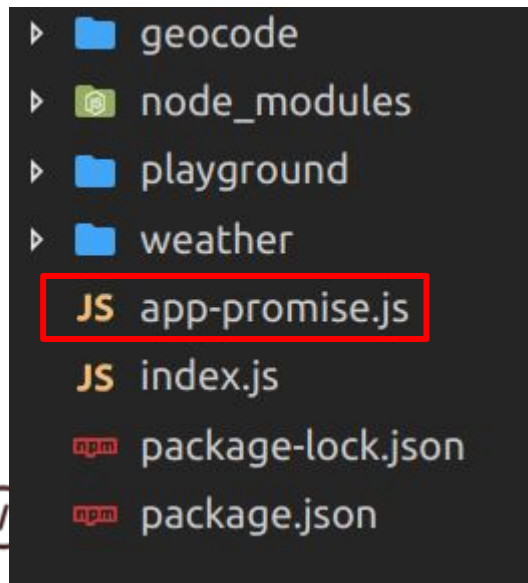
- `.then(...)` đầu tiên là phương thức của `add(1, 2)`
- `.then(...)` thứ 2 là phương thức của `add(res, 4)`
- `.catch(...)` là phương thức cho cả 2 lần gọi function `add(...)` trên.
- **Chú ý:** các bạn có thể thay `2` --> `'2'` hoặc `4` --> `'4'` để kiểm tra kết quả.

```
add(1, 2)
  .then(res => {
    console.log('The sum is: ' + res);
    return add(res, 4)
  })
  .then(res => console.log('The total sum is: ' + res))
  .catch(err => console.log(err));
```



# AXIOS

- Trở về bài toán, chúng ta sẽ áp dụng **promise** vào project
- Vấn đề: có những hàm hỗ trợ promise, có hàm không. Trong đó, **request** không hỗ trợ promise
- Do đó, chúng ta sử dụng **axios**
- Cài đặt: **npm install axios**
- Viết lại **weather app** bằng axios
- Tạo file **./app-promise.js**



# AXIOS

```
const yargs = require('yargs');  
const axios = require('axios');
```

```
const argv = yargs  
  .options({  
    a: {  
      demand: true,  
      alias: 'address',  
      describe: 'Enter your target address',  
      string: true  
    }  
  })  
  .help()  
  .alias('help', 'h')  
  .argv;
```

```
let encodedAddress = encodeURIComponent(argv.address);  
let geocodeUrl = `https://maps.googleapis.com/maps/api/geocode/json?  
key=[REDACTED]&address=${encodedAddress}`;
```



CYBERSOFT  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

# AXIOS

## Axios gọi geocode API

```
axios.get(geocodeUrl).then(res => {  
  if(res.data.status === 'ZERO_RESULTS'){  
    throw new Error('Unable to find that address');  
  }  
  console.log('The address: ' + res.data.results[0].formatted_address)  
}).catch(err => {  
  if(err.code === 'ENOTFOUND') {  
    console.log('Unable to connect to Google servers')  
  } else {  
    console.log(err.message)  
  }  
})
```

Tạo ra error mới và ném vào .catch(...)

Kết quả được in ra khi get thành công

Error khi không thể kết nối đến server

# AXIOS

## Axios gọi darksky API

```
console.log('The address: ' + res.data.results[0].formatted_address)
console.log("=====");

let lat = res.data.results[0].geometry.location.lat;
let lng = res.data.results[0].geometry.location.lng;
let darkskyUrl = `https://api.darksky.net/forecast/b8164e69c9f7fbc654f20d2d6381d1fc/${lat},${lng}`;
```

```
return axios.get(darkskyUrl);
```

Gọi API darksky API

```
.then(res => {
  console.log('Summary: ' + res.data.currently.summary);
  console.log('Icon: ' + res.data.currently.icon);
  console.log('Temperature: ' + res.data.currently.temperature);
})
```

Phương thức của  
axios.get(darkskyUrl)

```
.catch(err => {
  if (err.code === 'ENOTFOUND') {
```



# KẾT QUẢ CUỐI CÙNG

Run lệnh tại terminal:

```
node app-promise.js -a "459 su van hanh"
```

Thu được kết quả sau:

```
The address: 459 Sư Vạn Hạnh, Phường 12, Quận 10, Hồ Chí Minh, Vietnam  
=====  
Summary: Humid and Partly Cloudy  
Icon: partly-cloudy-day  
Temperature: 89.48
```



# REVIEW KIẾN THỨC

- Asynchronus
- Callback function
- Callstack và event loop
- Promise và Promise chain
- Google API, Weather API (darksky API)
- Package: request, axios, yargs





# HAPPY CODING :))

